Keywords in Python

1. In: The in keyword has two purposes:

* The in keyword is used to check if a value is present in a sequence (list, range, string etc.).
* The in keyword is also used to iterate through a sequence in a for loop:

Check if "banana" is present in the list:

fruits = ["apple", "banana", "cherry"]  
  
if "banana" in fruits:  
  print("yes")

Loop through a list and print the items:

fruits = ["apple", "banana", "cherry"]  
  
for x in fruits:  
  print(x)

1. With:

with statement is used to wrap the execution of a block of code within methods defined by the context manager.

Context manager is a class that implements \_\_enter\_\_ and \_\_exit\_\_ methods. Use of with statement ensures that the \_\_exit\_\_ method is called at the end of the nested block. This concept is similar to the use of try…finally block. Here, is an example.

with open('example.txt', 'w') as my\_file:

my\_file.write('Hello world!')

This example writes the text Hello world! to the file example.txt. File objects have \_\_enter\_\_ and \_\_exit\_\_ method defined within them, so they act as their own context manager.

First the \_\_enter\_\_ method is called, then the code within with statement is executed and finally the \_\_exit\_\_ method is called. \_\_exit\_\_ method is called even if there is an error. It basically closes the file stream.

1. Yield:

yield is used inside a function like a return statement. But yield returns a generator.

Generator is an iterator that generates one item at a time. A large list of values will take up a lot of memory. Generators are useful in this situation as it generates only one value at a time instead of storing all the values in memory. For example,

>>> g = (2\*\*x for x in range(100))

will create a generator g which generates powers of 2 up to the number two raised to the power 99. We can generate the numbers using the next() function as shown below.

>>> next(g)

1

>>> next(g)

2

>>> next(g)

4

>>> next(g)

8

>>> next(g)

16

And so on… This type of generator is returned by the yield statement from a function. Here is an example.

def generator():

for i in range(6):

yield i\*i

g = generator()

for i in g:

print(i)

**Output**

0

1

4

9

16

25

Here, the function generator() returns a generator that generates square of numbers from 0 to 5. This is printed in the for loop.

**Async keyword:**

Functions form the base of asynchronous programming. These async functions have async modifiers in their body. Here is an example of a general async function below:

When an async function is called, a Future is immediately returned and the body of the function is executed later. As the body of the async function is executed, the Future returned by the function call will be completed along with its result. In the above example, calling demo results in the Future.

Any functions you want to run asynchronously need to have the async modifier added to it. This modifier comes right after the function signature, like this:

|  |
| --- |
| void hello() async {    print('something exciting is going to happen here...');  } |

Await keyword:

Await expressions makes you write the asynchronous code almost as if it were synchronous. In general, an await expression has the form as given below:

void main() async {

  await hello();

  print('all done');

}

Typically, it is an asynchronous computation and is expected to evaluate to a Future. The await expressions evaluate the main function, and then suspends the currently running function until the result is ready–that is, until the Future has completed. The result of the await expression is the completion of the Future.

Q2) difference between print and return in python?

The print() function writes, i.e., "prints", a string or a number on the console. The return statement does not print out the value it returns when the function is called. It however causes the function to exit or terminate immediately, even if it is not the last statement of the function.

Functions that return values are sometimes called fruitful functions. In many other languages, a function that doesn’t return a value is called a procedure.

In the given code the value returned (that is 2) when function foo() is called is used in the function bar(). These return values are printed on console only when the print statements are used as shown below.

def foo():

   EX:

print("Hello from within foo")

    return 2

def bar():

    return 10\*foo()

print foo()

print bar()

O/P

Hello from within foo

2

Hello from within foo

20

We see that when foo() is called from bar(), 2 isn't written to the console. Instead it is used to calculate the value returned from bar().